Living Atlas

Project Alpha Prototype Report

Center for Environmental Research, Education, and Outreach (CEREO)

WASHINGTON STATE UNIVERSITY

Living Atlas Development Team

Long, Joshua

Kolb, Mitchell, William

Svetlik, Sierra Amelia

4/23/2023

I. Introduction

This document serves as an update of CEREO Living Atlas's progress on the Living Atlas app. It will cover the current progress of CLA on the Living Atlas app. Included in this document will be details on the project and team members, Living Atlas specifications and requirements, our approach to solving the given the issue, and the plans for testing the app. This document will also provide updates of the project and details of the progress made.

I.1. Project Introduction

CLA has been tasked with making an app that will be known as Living Atlas. The Living Atlas app will be able to collect environmental information about the Columbia River Basin of many different types, as well as having this information accessible by anyone. The ability to input information in the app will be restricted, but the ability to view the information will not be. The target groups for the app includes researchers, locals, tribes, and government officials. The end goal of this project is to have a repository of a wide variety of environmental data that can be accessed by anyone and uploaded and edited by anyone with permission.

I.2. Background and Related Work

The Center for Environmental Research, Education, and Outreach (CEREO) is interested in collecting information about the Columbia River Basin, but there is no good way to record and access the many different types of information that they are interested in. They want a wide variety of individuals, companies, and organizations to be able to input, edit, and access the information on the app.

The app can be compared to Zillow, but for environmental information. The user should be able to pull up a map and be able to see the information, and sort by types and categories of information.

Other websites that the app can be compared to include StoryMaps and a government data repository. StoryMaps has a map feature, and the data repository has a search feature. Both would be desirable in our app.

I.3. Project Overview

Living Atlas is a web application aimed at solving the problem of scattered and inaccessible environmental data. Our goal is to provide a central location for collecting, sharing, and accessing environmental data, making it available to a wide range of stakeholders including tribal communities, academic institutions, and government agencies.

The platform will allow for easy viewing by everyone but will require authentication for uploading data to ensure the validity and accuracy of the information being shared. Additionally, the platform will be able to connect to external sources, further expanding the range of environmental data available on the platform.

Living Atlas will offer the following key features and functionality:

1. Data Collection: A user-friendly interface for collecting and uploading environmental data from various sources, including individuals, institutions, and government agencies. The uploader will first be verified to ensure the data's accuracy and validity.
2. Data Management: An efficient and organized system for managing and storing the collected data, making it easily accessible and searchable. Users will be able to sort, filter, and analyze the data to gain valuable insights.
3. Data Sharing: A platform for sharing environmental data with the wider community, promoting collaboration, and improving access to data. The platform will be designed to be easily accessible and viewable by anyone.
4. External Data Connection: The ability to connect to external data sources, expanding the range of environmental data available on the platform.
5. Data Visualization: An interactive map-based interface for visualizing environmental data. The map-based interface will allow users to view data in a geographical context, providing valuable insights and promoting a deeper understanding of the environment.

Team Structure:

The Living Atlas project will be divided into two teams, each responsible for a specific aspect of the project. The clear division of roles and responsibilities between the Platform Team and Visualization Team is a key aspect of the project's success. This team structure helps to ensure the efficient and effective delivery of the Living Atlas project. Each team can focus on its specific tasks and collaborate effectively to achieve the overall goals of the project.

Living Atlas1: Platform Team

- The Platform Team will handle the data collection, management, and sharing aspects of the Living Atlas project. This includes creating a user-friendly interface for collecting data, an efficient system for managing data, a platform for sharing data, and the ability to connect to external data sources. The goal is to provide a comprehensive solution for collecting, managing, and sharing environmental data.

Living Atlas2: Visualization Team

- The Visualization Team will be responsible for creating the data visualization component of the Living Atlas project. They will design and develop interactive and user-friendly visualizations of the environmental data collected by the platform. The Visualization Team will work closely with the Platform Team to ensure that the visualizations are aligned with the platform's data management and sharing features. The ultimate goal is to make the environmental data easily accessible and understandable for the platform's users.

In conclusion, Living Atlas aims to make a significant impact by providing a central hub for environmental data, enabling better decision-making and promoting collaboration among a diverse group of stakeholders. We are committed to making this project a success and are excited to see the positive impact it will have on the environmental community.

I.4. Client and Stakeholder Identification and Preferences

Our Client is the Center for Environmental Research, Education, and Outreach (CEREO) team. CEREO seeks to apply innovative technologies and management tools to the ever-growing challenges of climate change and environmental sustainability. The CEREO team has a network of 350 faculty, staff, students, and outside industry leaders. They operate as a clearinghouse for a wide range of projects such as watershed management, tracking the nitrogen cycle, and studying urban socio-ecological systems. The subteam that we will be working with is the co-directors of CEREO, Dr. Jan Boll and Dr. Julie Padowski, and some additional core faculty/staff such as Dr. Hannah Haemmerli and principal assistant Jacqueline Richey McCabe.

The users of the Living Atlas website are the CEREO team and the National Science Foundation Traineeship Program which has partnered with CEREO. Other users include tribal communities, academic institutions, and government agencies. The client has stated that they intend to use this website to help share documents so the potential users are the future affiliates of the CEREO team. Possible other parties that relate to this project could be maintenance/servicers that host the website once it is operational and deployed. Cloud-based workers could also be included if when the website is deployed it is through a cloud service provider.

To allow for efficient communication, the CEREO team requests a web application that uses an interactive map and database to store and display information regarding rivers, watersheds, and communities in the larger pacific northwest region focusing around the Columbia river basin. This application will require the implementation of a well-structured and organized user interface and filtering system. Living Atlas team 1 will focus on the database and backend of the web application while Living Atlas team 2 will focus on the user interface and interactive map. This web application is intended to solve the problem that there is no one place to collect and share environmental data.

Lastly, the CEREO team requests that the web application be accessible to all users who wish to use it. Although it will be viewable to all, users who wish to contribute to the data shown on this web application will have to be approved by the CEREO team. The Living Atlas teams must build a web application with these preferences in mind.

II. Team Members - Bios and Project Roles

Joshua Long is a computer science major with a passion for software development. He is pursuing a Bachelor of Science in Computer Science and is eager to apply his technical skills to real-world problems. Joshua has experience with programming languages such as C#, JavaScript, and Python. He is dedicated to becoming a software developer and is eager to make a positive impact on the industry. Joshua is also the team lead of Living Atlas 1 which is responsible for handling the data collection, management, and sharing aspects of the Living Atlas project.

Mitchell Kolb is a senior majoring in computer science interested in machine learning and mobile/desktop application development. The languages Mitchell is most skilled in include C/C++, Python, HTML/CSS, and SQL. He has knowledge of technologies/frameworks such as Git, Flask, Angular, PyTorch, and Pandas. Mitchell's responsibilities include assisting his team with the creation of a database and backend for the Living Atlas website.

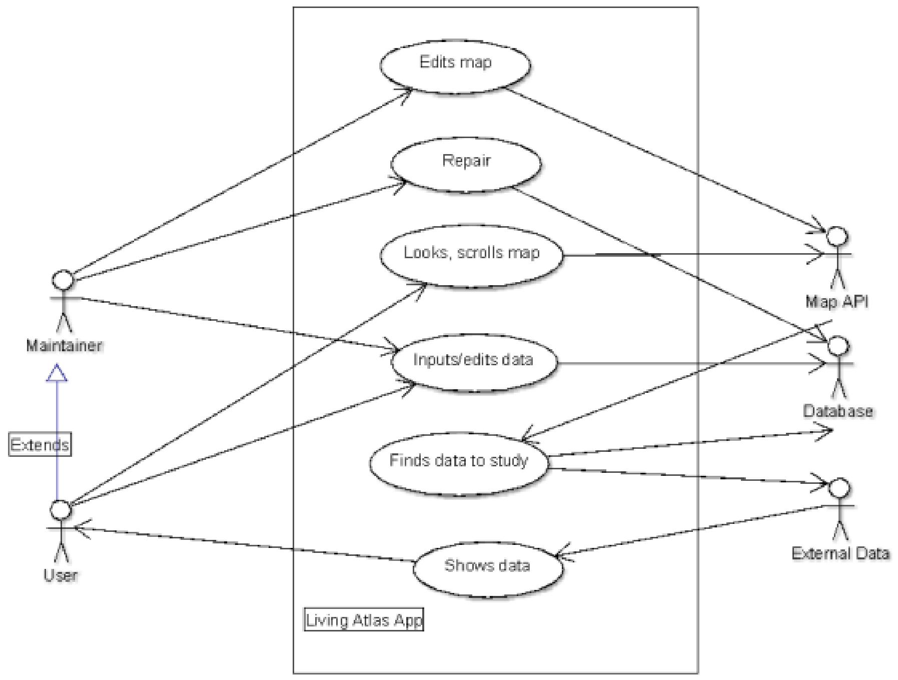
Sierra Svetlik is a computer science student who loves to code. She is currently pursuing a Bachelor of Computer Science at Washington State University. She has experience with C/C++, Python, and SQL. Sierra also has some experience with C#, Haskell, and JavaScript. She has a goal of developing video games but would be happy to apply her knowledge anywhere to make a positive impact on the world. She is part of Living Atlas team 1.

III. Project Requirements

This document serves as an update of CEREO Living Atlas's progress on the Living Atlas app. It will cover the current progress of CLA on the Living Atlas app. Included in this document will be details on the project and team members, Living Atlas specifications and requirements, our approach to solving the given the issue, and the plans for testing the app. This document will also provide updates of the project and details of the progress made.

III.1. Use Cases

CLA has been tasked with making an app that will be known as Living Atlas. The Living Atlas app will be able to collect environmental information about the Columbia River Basin of many different types, as well as having this information accessible by anyone. The ability to input information in the app will be restricted, but the ability to view the information will not be. The target groups for the app includes researchers, locals, tribes, and government officials. The end goal of this project is to have a repository of a wide variety of environmental data that can be accessed by anyone and uploaded and edited by anyone with permission.



Story: A student wants to write a paper on nitrogen in the Columbia River Basin. To find information about the presence of nitrogen in the river, the student goes to the Living Atlas application. After looking around the map a bit, they find a spot of interest and look up nitrogen using either a search bar feature or a category menu. Papers and other data related to the search or category appear in a menu next to the map, these results being tied to the area of the map that the student is looking at. The results were found using categories tagged to them or finding the word “nitrogen” in their title. The student is then able to open the papers or other data to use in their research, and can be confident in their validity, as only verified users and data can be uploaded to the application.

Source: CEREO Staff Member

Story: A researcher is studying the clusters of salmon in the Columbia River Basin. After collecting numbers for different clusters of salmon throughout a section of the river, they want to upload it to the Living Atlas. They first log into the application, then pinpoint an area where they found one of the clusters of salmon. They choose to upload data to that coordinate, start filling in a template for their data, including various different kinds of information about their data. This info includes their name, data of collection/upload, among other types of info (to be determined). They then upload their research. Next, the file is given an ID so that it can be uniquely recognized inside the database and is tagged with various categories that it can be found under. This process is repeated for all the different coordinates that the researcher found cluster of salmon at.

Source: CEREO Staff Member

III.2. Functional Requirements

1. Data collection:

- The system must provide a user-friendly interface for collecting and uploading environmental data from various sources, including individuals, institutions, and government agencies.
- The system must verify the accuracy and validity of the uploaded data through a verification process.
- The system must provide feedback to the uploader in case of invalid or inaccurate data.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for collecting accurate and reliable environmental data.

Priority: Priority Level 0: Essential and required functionality

2. Data management:

- The system must provide efficient and organized storage for the collected data, allowing easy accessibility and searchability.

- The system must support sorting, filtering, and analysis of the data to gain valuable insights.
- The system must support data versioning to keep track of changes and revisions made to the data over time.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for effectively managing and utilizing the collected environmental data.

Priority: Priority Level 0: Essential and required functionality

3. Data sharing:

- The platform must provide an interface for sharing environmental data with the wider community, promoting collaboration, and improving access to data.
- The platform must support secure and controlled sharing of data with specific individuals or groups.
- The platform must provide a notification system to inform users of new or updated data.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for promoting collaboration and providing valuable insights into the environment.

Priority: Priority Level 0: Essential and required functionality

4. External data connection:

- The system must support connections to external data sources to expand the range of environmental data available on the platform.
- The system must support various data formats and protocols for importing data from external sources.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for providing a comprehensive view of the environment and enabling users to gain valuable insights.

Priority: Priority Level 1: Desirable functionality

5. Data visualization:

- The system must provide an interactive map-based interface for visualizing environmental data in a geographical context.
- The map-based interface must support various map layers and overlay options for viewing data in different ways.
- The interface must allow users to filter and customize the displayed data based on various parameters.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for providing a comprehensive view of the environment and enabling users to gain valuable insights.

Priority: Priority Level 0: Essential and required functionality

III.3. Non-Functional Requirement

Performance: The Living Atlas website should load quickly and respond to user interactions promptly, even during peak traffic periods. The site should also be scalable and able to handle increases in traffic without slowing down.

Usability: The site's user interface should be intuitive and easy to navigate, with a clear hierarchy of information and consistent design elements throughout the site. The site should also be accessible to users with disabilities, with features like alt text for images and keyboard navigation. It should also be inclusive for users who are apart of native tribes in the pacific northwest region.

Security: The Living Atlas website should protect users' personal information, including passwords, uploaded/linked files, and personal data. The site should use encryption and other security measures to prevent unauthorized access and data breaches.

Compatibility: The site should be compatible with a wide range of browsers, devices, and operating systems, so that users can access it from wherever they are and on whatever device they choose.

Reliability: The Living Atlas website should be available and responsive at all times, with mechanisms in place to handle outages and downtime. The site should also have backup and recovery measures in place to ensure that data is not lost in case of a failure.

Maintainability: Once the Living Atlas website is deployed it will be maintained by WSU technical department. Therefore, it should be built with maintainability in mind, with clean and well-documented code, and should be easy to modify or update as needed.

Efficiency: The Living Atlas website should be optimized for performance and efficency to reduce page load times and minimize server load. The site should also be designed with energy efficiency in mind, using best practices for web development to minimize energy consumption.

Data Management: The site should be able to manage large volumes of data, such as information relating to watersheds, rivers, and communities, with high availability, scalability, and reliability. It should also have robust data backup and recovery mechanisms in place.

Searchability: The site's search functionality should be efficient, accurate, and fast. It should return relevant results, provide filtering options, and be able to handle large volumes of data.

IV. Solution Approach

IV.1. Front-End

Subsystem Decomposition:

Header Component:

1) Description: The Header component displays the logo and navigation links, Search bar and filtering button, and Upload Button.
2) Concepts and Algorithms Generated: The Header component utilizes concepts such as UI design and user experience, as well as algorithms for rendering and positioning UI elements.

Upload Form Component:

1) Description: The Upload Form component provides users with a way to upload data to the application, including selecting a file to upload and entering metadata about the data. The Form must have eleven variables, name, email, funding, organization, title, link, description, tags, latitude, longitude, and file.
2) Concepts and Algorithms Generated: The Upload Form component utilizes concepts such as form validation and data modeling, as well as algorithms for handling file uploads and data storage. Also, it must have the ability to send a POST request to the Backend.

Content Area Component:

1) Description: The Content Area component is responsible for fetching data from the backend using a GET request and rendering the Card components based on the data retrieved.
2) Concepts and Algorithms Generated: The Content Area component utilizes concepts such as asynchronous programming and data fetching, as well as algorithms for parsing and processing data retrieved from the backend.

Card Component:

1) Description: The Card component displays information about a specific piece of data, including name, email, funding, organization, title, link, description, tags, latitude, longitude, and file.
2) Concepts and Algorithms Generated: The Card component utilizes concepts such as data modeling and UI design, as well as algorithms for rendering UI elements based on data passed in.

IV.2. Back-End

The Backend is made up of restful endpoints that allow the frontend to access the data from the database in a clean and concise matter. All endpoints in this project are made using FastAPI with python.

Endpoint Decomposition:

Card Endpoint:

- Description: This endpoint is a GET request from the frontend that is for when the dynamic cards are being displayed for the main page of the website. This endpoint is only for the preview of the cards, so the information returned is not the complete amount that is available. This endpoint returns the Research data point titles and descriptions so they can be displayed.

Card Details Endpoint:

- Description: This endpoint will be used when the user clicks into a card on the right side of the website. The front end will send over the card title so that it can be matched with the correct row in the database. All research point titles are unique at this stage in the project. The endpoint will return the entire list of details regarding that research point. The details include username of the user who posted the point along with their email address, the funding, organization, link, and description of the project, along with any tags that are associated with the point. Lastly the location points of the project are returned.

Upload Form Endpoint:

- Description: This endpoint will be used when the user wants to create their own research data point to be displayed on our site. Research data points include this data that will be sent over in a form from the front end to the back end. Username, email, title, description, date, funding, organization, links, and all tags. After that the backend will query the database using INSERT to input the data appropriately.

Get Research Data Endpoint:

- Description: This endpoint returns the row or column of available data when given a row title or column label. This will be used when displaying small details on the map when we connect Living Atlas group 1 and group 2 projects together.

Update Research Data Endpoint:

- Description: This endpoint will receive a title or did (data id) and specified data type that will be updated from the front end and use a PUT request to update the database column and row with the new data.

Delete Research Data Endpoint:

- Description: This endpoint will receive a title or did (data id) from the front end and use a DELETE request to remove a research data point from the database to no longer be displayed or stored. This endpoint will return a confirmation notice to the front end to ensure that all sides know that the deleted data has been successfully removed.

Get Research Data Location Endpoint:

- Description: This endpoint will receive a title or did (data id) from the front end and use a GET request to retrieve the location data of a particular point from the database and return it to the front end to be displayed on the map.

IV.3. Database

The database will utilize various tables to store the information efficiently.

Schema breakdown:

Users table:

The users table will store information about the users of the app, with primary key uid (user id). The users table will also store the username, the password in a hashed form, the email, and the salt for the password.

Data Entry table:

The data entry table will store some of the info about the data uploaded to the app, with primary key did (data id) and foreign key uid (user id). The data entry table will also store the data the data was uploaded, the title of the data, a link to the data, and optionally the latitude, longitude, funding, and organization.

County table:

The county table will store counties associated with the data uploaded to the app, with the tuples acting as the keys. The table will store the did (data id) and counties. The did also acts as the foreign key.

Watershed table:

The watershed table will store watersheds associated with data uploaded to the app, with the tuples acting as the keys. The table will store the did (data id) and watersheds. The did also acts as the foreign key.

GMA table:

The GMA table will store GMAs associated with data uploaded to the app, with tuples acting as the keys. The table will store the did (data id) and GMAs. The did also acts as the foreign key.

City table:

The city table will store cities associated with the data uploaded to the app, with tuples acting as the keys. The table will store the did (data id) and cities. The did also acts as the foreign key.

State table:

The state table will store states associated with the data uploaded to the app, with tuples acting as the keys. The table will store the did (data id) and states. The did also acts as the foreign key.

Water Quantity table:

The water quantity table will store information about data regarding water quantity, with the did (data id) acting as the primary key, as well as the foreign key. The table will also store booleans referring to whether the information is associated with uses (uses has its own table), precipitation, streamflow, drought index, and reservoir volume.

Use table:

The use table will store information about water usage associated with the data uploaded to the app, with the did (data id) acting the primary, as well as the foreign key. The table will also store booleans referring to whether the information is associated with municipal, agricultural, and/or industrial use.

Water Quality table:

The water quality table will store information about data regarding water quality, with the did (data id) acting as the primary key, as well as the foreign key. The table will also store booleans referring to whether the information is associated with sediment, nutrients (nutrients has its own table), temperature, fecal coliform, toxins, and dissolved oxygen.

Nutrient table:

The nutrient table will store information about nutrients associated with the data uploaded to the app, with the did (data id) acting the primary, as well as the foreign key. The table will also store booleans referring to whether the information is associated with nitrogen, phosphorus, and/or carbon.

Discipline table:

The discipline table will store information about data regarding various disciplines, with the did (data id) acting as the primary key, as well as the foreign key. The table will also store booleans referring to whether the information is associated with biology, ecology, chemistry, physics, socioeconomics, and jurisdictions.

Source table:

The source table will store information about the source of the data, with the did (data id) acting as the primary key, as well as the foreign key. The table will also store booleans referring to whether the information is sourced from academia, the government, tribes, NGO, and others.

V. Test Plan

V.1. Overview

Test Objectives and Schedule:

The test objectives for the Living Atlas application include testing the primary functions of the application. The testing schedule includes conducting functional testing during the development phase and conducting acceptance testing with sponsors and clients during the staging phase. Additionally, non-functional requirements such as performance testing will be conducted using appropriate testing strategies and tools. The testing process will be iterative, with bugs and issues being reported and addressed throughout the development and staging phases. Overall, the testing objectives and schedule aim to ensure the quality and functionality of the Living Atlas application.

Testing strategies and tools for functional requirement:

For testing the functional requirements of our Living Atlas application, we will be using a combination of manual and automated testing. Manual testing will involve testing each individual component and feature of the application to ensure they are functioning correctly. Automated testing will be carried out using Selenium for the front end, which will allow for efficient and accurate testing of the application's functionality across different browsers and platforms. Postman will be used for the backend to test accuracy of the returned data.

Testing strategies and tools for non-functional requirement:

For testing non-functional requirements such as performance and security, we will be using tools such as JMeter, Loadster. These tools will allow us to simulate real-world scenarios and identify any bottlenecks or vulnerabilities in the application. Additionally, the front end will be setting up alerts and monitoring tools to keep track of the application's performance and detect any issues in real-time. The back end will be using a program called Locust which can simulate user behavior and test the performance of the FastAPI endpoints. It can do this by being able to generate thousands of concurrent users and measure the response time throughout the API.

V.2. Front-End

Functional Testing:

For the functional testing of our Living Atlas application, we will be focusing on testing the primary/core functions of the application. For the Front-End this includes testing the ability to upload and download data, search for data by keyword, displaying data by cards. We will be using the following test cases to ensure that the core functions of the application are working as expected.

1. Test the ability to upload data to the Living Atlas application

Steps	Action	Expected Response	Pass/Fail	comments

1	Navigate to the upload page	Successfully navigate to the upload page when press upload button		
2	Select a file to upload	Successfully select a file to upload when press upload button		
3	Verify that the file is uploaded successfully	Successfully receive confirmation that the file was uploaded successfully.		

2. Test the ability to download data from the Living Atlas application

Steps	Action	Expected Response	Pass / Fail	Comments
1	Navigate to the download page	Press the card and make it show the download button		
2	Search for the desired data	Able to show the data description		
3	Select the data to download	Able to press the download button		
4	Verify that the data is downloaded successfully	Download the data		

3. Test the ability to search for data by keyword Steps:

Steps	Action	Expected Response	Pass/Fail	comments
1	Navigate to the search page	See the search bar		
2	Enter a keyword to search for	See the text you entered		
3	Verify that the search results	See the text you enter is correct		

	are returned correctly			

4. Test the ability to displaying data by cards properly

Steps	Action	Expected Response	Pass/Fail	comments
1	Use a mock data set with known values to ensure the cards are rendered correctly	Store mock data in the database		
2	Open the application	Should be able to launch application		
3	Verify that all card information is displayed accurately and in the correct format.	Cards should show all the content of the moch data		

V.3. Back-End

Functional Testing:

For the functional testing of our Living Atlas application, we will be focusing on testing the primary/core functions of the application. For the Back End this includes testing the ability to take in form data from the front end and return the correct data relating to it and to return data from various requests.

1. Test the ability to upload data to the Living Atlas application database

Steps	Action	Expected Response	Pass/Fail	comments
1	Be given a form data type that contains all inputted data for the new research point	All data is assigned/converted to variables that can be used within the endpoint. “Converted Complete” will be printed to the terminal		

2	Perform POST request by querying the database with the INSERT statement	"Completed POST" will be printed to the terminal and returned to the front end. "Failed to insert" will be returned otherwise		

2. Test the ability to return data from the Living Atlas application database

Steps	Action	Expected Response	Pass/Fail	comments
1	Be given a title or UID from the frontend	The data is assigned/converted to variables that can be used within the endpoint. "Converted Complete" will be printed to the terminal		
2	Perform GET request by querying the database with the SELECT statement	"Completed GET" will be printed to the terminal and returned to the front end. "Failed to insert" will be returned otherwise		

3. Test the ability to delete data from the Living Atlas application database

Steps	Action	Expected Response	Pass/Fail	comments

1	Be given a title or UID from the frontend	The data is assigned/converted to variables that can be used within the endpoint. “Converted Complete” will be printed to the terminal		
2	Perform DELETE request by querying the database with the DELETE statement	“Completed DELETE” will be printed to the terminal and returned to the front end. “Failed to insert” will be returned otherwise		

Nonfunctional Testing:

For the nonfunctional testing of our Living Atlas application, we will be focusing on testing the primary/core functions of the application. For the Back End this includes testing the performance of the endpoints to ensure that as this project scales up in the number of users it can still perform at the same efficiency and accuracy.

1. Test the ability to service more than one user who are using Living Atlas application

Steps	Action	Expected Response	Pass/Fail	comments
1	Async functionality on endpoints allows multiple users to request from the same endpoint.	Data is returned to both users		

VI. Staging and Deployment

VI.1. Front-End

Staging: To stage the software for acceptance testing, we plan to create a separate staging environment that will be a replica of the production environment. This environment will be hosted on a separate server and will contain all the necessary infrastructure and dependencies required for the application to run.

Deployment: For the production deployment of the application, we plan to use the free tier of cloud-based platform, such as AWS or Google Cloud, to host the application.

Development and Deployment Environment: The development environment for the application is based on React, and we have set up a local development environment on our development machines. We use code editors like Visual Studio Code, and package managers like npm to manage dependencies and run the application locally. We use Git for version control, and we have set up a remote Git repository to manage the source code for the application. For the Deployment Environment we might have to use a Docker container or similar package that can be deployed to the production environment.

VI.2. Back-End

Staging: To stage a FastAPI application, we need to create a separate environment where the application can be tested and validated before deploying it to production. The staging environment should closely resemble the production environment, including the operating system, dependencies, and network configuration.

To set up a staging environment, we can use tools like Docker to create a containerized version of the application that can be easily deployed and tested on any machine. We can also use continuous integration and deployment (CI/CD) tools like Jenkins, TravisCI, or CircleCI to automate the deployment and testing process.

Deployment: When it comes to deploying a FastAPI application to production, we have several options depending on the specific needs and requirements of the project. Here are some common deployment strategies:

I. Cloud-based deployment: We can use cloud platforms like AWS, Google Cloud, or Azure to deploy the FastAPI application in a scalable and cost-effective manner. These platforms offer a wide range of tools and services for managing infrastructure, deploying applications, and monitoring performance.

II. Self-hosted deployment: Alternatively, we can deploy the application on a dedicated server or virtual machine using tools like Docker, Kubernetes, or Ansible. This gives us more control over the infrastructure and allows us to customize the deployment environment to our specific needs.

Development and Deployment Environment: In terms of the development environment, we need to set up a local environment where we can develop and test the FastAPI application before deploying it to staging or production. This environment should include the necessary dependencies and tools, such as Python, the FastAPI framework, and a database.

We can use tools like virtual environments or Docker to create a reproducible development environment that closely matches the staging and production environments. This helps to ensure that the application works as expected across all stages of the development lifecycle.

Planned deployment environment: The planned deployment environment should be designed to meet the specific requirements of the project, including performance, scalability, and security. This might involve setting up load balancers, auto-scaling groups, or implementing security measures like SSL/TLS encryption or access controls.

VI.3. Database

Staging: To stage the SQL database hosted on ElephantSQL for acceptance testing, we can create a separate testing environment that replicates the production environment. This environment should contain a copy of the production data and all the necessary database infrastructure and dependencies required for the application to run. We can use database migration tools like Alembic or Flyway to manage schema changes and seed data for the testing environment. For the production deployment of the database, we plan to use the paid tier of ElephantSQL or a similar database-as-a-service provider. This will ensure that the database is highly available, scalable, and secure.

Deployment: The development environment for the SQL database can be set up locally using tools like SQL clients such as pgAdmin or DBeaver. We can also use ORMs like SQLAlchemy to abstract the database operations and make it easier to write and test database code. We use Git for version control and have set up a remote Git repository to manage the database schema and data scripts.

Development and Deployment Environment: For the deployment environment, we plan to use Docker to containerize the database and its dependencies. This will ensure that the deployment process is reproducible, and that the database can be easily moved between different environments. We can also use CI/CD tools like Jenkins, TravisCI, or CircleCI to automate the deployment and testing process.

III. Alpha Prototype Description

a. Front-End

Header Component:

1) Create a new React component named Header
2) Inside the Header component, use the HTML to create the links and buttons, then add the upload form element(Form modal component).

Upload Form Component:

1) Create a new React component named UploadForm(Form modal component).

2) Inside the UploadForm component, create a form element with the required fields for data upload (name, email, funding, organization, title, link, description, tags, latitude, longitude, and file)
3) Use React's state to store form input data and update the state when the user inputs data
4) Add a submit handler to the form that sends a POST request to the backend API using Axios

Content Area Component:

1) Create a new React component named ContentArea
2) Inside the ContentArea component, use the useEffect hook to fetch data from the backend API using the Axios GET request.
3) Use React's state to store the fetched data and update the state when the data is fetched
4) Pass the fetched data as props to the Card component using the map function.

Card Component:

1) Create a new React component named Card
2) Inside the Card component, use the HTML div element to display the data passed in as props (name, email, funding, organization, title, link, description, tags, latitude, longitude, and file)
3) Also add a download button to enable downloading the file

b. Back-End

Endpoint Decomposition:

Card Endpoint:

This is implemented as of the writing of this document and returns the title and description of multiple research data points to be displayed on the cards that show on the right side of the main page of the website.

Card Details Endpoint:

This is implemented as of the writing of this document and this endpoint will be used when the user clicks into a card on the right side of the website. The front end will send over the card title so that it can be matched with the correct row in the database. All research point titles are unique at this stage in the project. The endpoint will return the entire list of details regarding that research point. The details include username of the user who posted the point along with their email address, the funding, organization, link, and description of the project, along with any tags that are associated with the point. Lastly the location points of the project are returned.

Upload Form Endpoint:

This is implemented as of the writing of this document and this endpoint will be used when the user wants to create their own research data point to be displayed on our site. Research data points include this data that will be sent over in a form from the front end to the back end. Username, email, title, description, date, funding, organization, links, and all tags. After that the backend will query the database using INSERT to input the data appropriately.

Get Research Data Endpoint:

This is implemented as of the writing of this document and this endpoint returns the row or column of available data when given a row title or column label. This will be used when displaying small details on the map when we connect Living Atlas group 1 and group 2 projects together.

Update Research Data Endpoint:

This feature has been coded but not added into the main codebase of the project because of issues that occur when dealing with data over many tables and with duplicate entries. This endpoint will receive a title or did (data id) and specified data type that will be updated from the front end and use a PUT request to update the database column and row with the new data.

Delete Research Data Endpoint:

This is implemented as of the writing of this document and this endpoint will receive a title or did (data id) from the front end and use a DELETE request to remove a research data point from the database to no longer be displayed or stored. This endpoint will return a confirmation notice to the front end to ensure that all sides know that the deleted data has been successfully removed.

Get Research Data Location Endpoint:

This feature hasn’t been implemented yet because we have not had a use for it given the fact that we have not combined our codebase with the other Living Atlas Groups work because they are work on the map features and this will come in handy when we have to get that to work with the data in the database. This endpoint will receive a title or did (data id) from the front end and use a GET request to retrieve the location data of a particular point from the database and return it to the front end to be displayed on the map.

c. Database

The database will utilize various tables to store the information efficiently. We will be using ElephantSQL for the remote database, PostgreSQL for managing the database, and psycopg2 for accessing the database from the code.

Schema breakdown:

Users table:

The users table stores information about the users of the app, with primary key uid (user id). The users table will also store the username, the password in a hashed form, the email, and the salt for the password.

Data table:

The data table stores information about the data that users upload to the app, with primary key did (data id) and foreign key uid (user id). The data table will also store the latitude, longitude, date, description, link, organization, funding, and title.

Tags table:

The tags table stores information about the categories of the data uploaded to the app, with the did (data id) and tag tuples acting as primary keys and the did also acting as the foreign key.

IV. Alpha Prototype Demonstration

a. Front-End

In the alpha prototype of Living Atlas, we have created a basic layout for the website, with a navigation menu, header, and footer. We have also implemented a card that uses a GET request to dynamically fetch data from the backend and display it in the proper format. Additionally, we have implemented an upload form that uses a POST request to send data to the backend for storage and processing. These features demonstrate the core functionality of Living Atlas and showcase the potential of the application to become a powerful tool for managing and visualizing environmental data.

b. Back-End

The back end for the Living Atlas website involves creating and managing APIs that handle data and logic on the server side of an application. Endpoints are a key concept in this process, they define the various operations that the API can perform and specify the expected input and output formats. I have been using a framework called Fast API which allows me to write these endpoints. Endpoints are created by defining functions that are decorated with specific HTTP methods such as GET, POST, PUT, DELETE, etc. and a URL path. These functions handle incoming requests by extracting data from the request body or query parameters, performing any necessary processing or validation or querying the database that we have, and returning a response in the form of JSON or other formats to the data can appear on the front ed portion we have also been developing.

c. Database

The database currently stores, within a user's table, the username, the password in a hashed form, the email, as well as an automatically generated salt and unique user ID. The database also contains a data table that stores a unique data ID, a link to the data, the date that the data was uploaded to the database, a title for the data, a description of the data, and the user ID of the user that uploaded the data. It can also store a latitude and longitude associated with the data, the source of funding, and the organization that the user associates with. There is also a tags table that stores, in each row, a data ID and a tag associated with the data. The tag represents a category that the user thinks is appropriate for the data, and there is one row for each data and tag grouping.

V. Future Work

a. Front-End

In future work, We plan to enhance the functionality of Living Atlas by enabling user login, which will allow users to securely access and manage their data. We also plan to enable file handling, which will allow users to upload and download data files in a variety of formats. In addition, we will enable filtering, which will enable users to sort and search data based on specific criteria. Finally, we plan to integrate a visualization map, which will provide users with an interactive map-based view of the data.

b. Back-End

For the future of the back end, we plan on creating more endpoints to keep up with the new changes that come with the development of the front end once we merge the two groups together. We will also assist more in the development of the tables in the database because we have learned that having an extremely clear idea of the database and all the data that it must serve will allow us to properly create endpoints that complete the task at hand. We will also look into using other API's to help complete the new features that the clients have mentioned in recent meetings, so we don't have to reinvent the wheel.

c. Database

For the future of the database, we plan to change the tables according to the solution approach. The new tables will be County, Watershed, GMA, City, State, Water Quantity, Use, Water Quality, Nutrient, Discipline, and Source. Adding these tables and dropping the Tags table will allow for better efficiency of the database and keep the tables to a minimal size when the number of users of the app increases. The Data table will be renamed to Data Entry in order to be clearer in its use.

VI. Repository information

a. Main Branch

a) Repo URL

https://github.com/WSUCapstoneS2023/LivingAtlas1

b) Last commit information

(Date) 4/18/2023

(hash) 5ede17f - joshmainac,

(URL) https://github.com/WSUCapstoneS2023/LivingAtlas1/commit/5ede17f

(Description)

Merge pull request #34 from WSUCapstoneS2023/josh#33-cards

Cards enable to send GET request from backend, and map data to card.

c) Contributor

Joshua Long - joshmainac

Kolb, Mitchell, William - Mitchell-kolb

Svetlik, Sierra Amelia - SierraSv

VII. Glossary, Reference & Appendices

Axios: Axios is a Javascript library used for making HTTP requests. It provides an easy-to-use interface for making requests to a server and handling the responses.

CLA: CLA is short for CEREO Living Atlas

ElephantSQL: ElephantSQL is a database hosting service that sells databases for customers to use remotely, instead of having all the information stored on local devices.

PostgreSQL: PostgreSQL is a database management service that allows the user to create and modify tables, as well as manage usage of the tables and many other details.

Psycopg2: Psycopg2 is a python package for accessing and modifying PostgreSQL databases.

React: React is a popular Javascript framework used for building user interfaces. It provides a declarative syntax for creating components that manage their own state and render based on that state. React uses a Virtual DOM to efficiently update the UI and provides a way to modularize code through the use of components.

Did: Data id

Uid: User id

FastAPI: The framework that backend will be using to develop its endpoints to be